秘密計算によるプライバシー保護勾配ブースト木

三品 気吹[1,a)]　濱田 浩気[1]　五十嵐 大[1]　菊池 亮[1]

概要：秘密計算とはデータを暗号化したまま計算する技術である．そのためプライバシーを保護したまま安全にデータ分析を行う方法として注目されており，中でも近年のデータ分析手法の主流である「機械学習」を秘密計算上で実現する研究は活発に行われている．機械学習手法の中でも特に有名なものとして決定木がある．また，決定木に対してアンサンブル学習手法の一つである「ブースティング」を適用した高精度な学習を実現する手法として「勾配ブースト木 (GBDT)」があり，高精度さや学習データの前処理の容易さから，深層学習と並んで近年よく用いられている．本稿では，勾配ブースト木を秘密計算上で行うアルゴリズムを提案・実装する．提案手法では，学習データや木の構造，各ノードでの分岐条件等を秘匿したまま勾配ブースト木の学習・予測行う．Boston データセットを用いた実験では，秘密計算ディープラーニングと比較して，40 分の 1 の時間で同等以上の予測性能を得られた．

キーワード：秘密計算，機械学習，決定木，勾配ブースト木

Privacy Preserving Gradient Boosted Decision Trees in Secure Computation

Ibuki Mishina[1,a)]　Koki Hamada[1]　Dai Ikarashi[1]　Ryo Kikuchi[1]

Abstract: Secure computation is a technique for computing while keeping data encrypted. Therefore, it is attracting attention as a method to analyze data securely while protecting privacy, and research to realize various machine learning methods on secret computation is actively being conducted. One of the most famous machine learning methods is the decision tree. Gradient Boosted Decision Trees (GBDT) is a method to achieve highly accurate learning by applying "boosting", an ensemble learning method, to decision trees, and has been used frequently in recent years along with deep learning due to its high accuracy and ease of preprocessing of training data. In this paper, we propose and implement an algorithm for performing gradient boosted trees on secure computation. In our experiments on the Boston dataset, we obtained the same or better prediction performance in 1/40th of the time compared with deep learning on secure computation.

Keywords: Secure Computation, Machine Learning, Decision Tree, Gradient-Boosted Decision Trees

1. はじめに

秘密計算とはデータを暗号化したまま計算する技術である．秘密計算を用いることで，企業の重要な情報や顧客情報を安全に守ったままデータ分析等に利用できるため，様々な統計手法や機械学習手法を秘密計算上で実現するための研究がなされている．特に秘密計算上での機械学習 (秘密計算 AI) に関する研究は近年非常に活発である．

機械学習には線形回帰，決定木，深層学習など様々な手法が存在し，新たな手法も次々と提案されている．機械学習で最も有名な手法はディープラーニングだが，近年では「勾配ブースト木 (Gradient Boosted Decision Trees)」などの，ディープラーニングより更に新しい手法もよく用いられるようになってきている．

秘密計算 AI の研究における重要な課題の一つは，線形回帰やロジスティック回帰といたデータ分析の領域で昔からよく用いられている手法を秘密計算上でもできるように

[1] NTT 社会情報研究所
NTT Social Informatics Laboratories
[a)] ibuki.mishina.br@hco.ntt.co.jp

網羅していくことだが，それと同時に，次々と新しく登場する機械学習手法を速やかに秘密計算上で実現することも重要な課題だと考えられる．

筆者らはこれまでにロジスティック回帰 [9]，線形回帰 [12]，ディープラーニング [7]，医療統計 [11][8] など多数の機械学習や高度な統計手法を秘密計算上で実現し，また一般的なデータ分析者にも使い易い秘密計算 AI のインターフェースの提案 [10] なども行うことで，秘密計算 AI の機能拡充に取り組んできた．本稿では秘密計算 AI のステップアップを目指し，最先端の機械学習手法である「勾配ブースト木 (Gradient Boosted Decision Trees)」[2] を秘密計算上で実現することに取り組む．勾配ブースト木は非常に高い予測精度を得られる手法として，近年よく用いられている．

1.1 関連研究

学習データのプライバシーを保護した状態で勾配ブースト木を行う方法を提案している論文としては LLS2019[4] がある．LLS2019 は federated learning[5] を用いてプライバシー保護勾配ブースト木を実現している．federated learning[5] は，学習データを保有する複数の参加者達が各々のローカル環境で学習を行ったのち，それらの情報を集約サーバで集約してモデル更新するという手法である．学習の参加者達は自分のデータを外に出す必要がないためプライバシーを守ることができる．

federated learning は学習への参加者が多い場合などに有用な手法だが，federated learning によって得られる学習結果と，データを 1 か所に集めて行う一般的な方法による学習結果は異なったり，参加者が少ない場合には集約後の学習結果から他者のデータを推測できる可能性があるため，必ずしも最適な手法ではない．例えばデータを一か所に集めてから学習を行う必要がある，もしくは元々データが一か所にある場合や，計算途中の値を復元しない高い安全性を求めるような場合には，最初から最後まで学習データを暗号化したまま処理できる秘密計算が適している．秘密計算であれば，学習データが暗号化されているだけでなく，計算途中の値も暗号化されているため，元データの推測に繋がるようなデータが漏れてしまう心配も無い．

1.1.1 課題

従来研究の課題は以下であると考える．

- データを暗号化したまま，勾配ブースト木の学習や予測を行う手法が無い

勾配ブースト木は平文でのデータ分析の領域でも非常に新しい手法であるため，まだプライバシー保護データ分析の領域では研究が進んでいないが，最先端の分析手法を秘密計算でもできるようにすることは重要である．特に勾配ブースト木はディープラーニングよりも高い精度が得やすいことや，決定木をベースとするために説明変数のスケールを気にする必要がないといったメリットがあるため優先度が高い．そこ本稿では，学習データや計算途中の値も全て暗号化したまま勾配ブースト木の学習・予測を行う「秘密計算勾配ブースト木」を提案・実装する．

1.2 本稿の貢献

本稿の貢献は以下である．

- 秘密計算による勾配ブースト木の学習・予測を実用的な処理速度で実現した
- 非常に高い予測精度を持つ AI を秘密計算上でも作れるようになった

秘密計算ディープラーニング [6] では 640 秒かかって得られた予測精度よりも高い精度を，本稿で提案する秘密計算勾配ブースト木では約 16 秒で達成した．

2. 準備

2.1 決定木

決定木はよく用いられる基本的な機械学習手法の一つであり，回帰分析にもクラス分類にも利用できる．決定木という名前の通り，根，枝，葉からなる木のような形をしたモデルである．図 1 は，年齢，体重，体脂肪などの特徴量から，ダイエットが必要か不要かを判断する決定木の例である．どのような条件でデータを分類したかが分かりやすいため，実用的なデータ分析手法として非常によく用いられる．

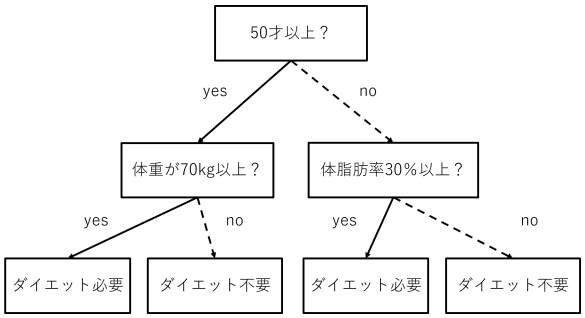

図 1: 決定木のイメージ図

決定木の学習では，各枝でどの特徴量に注目して，閾値をどのように設定したらデータを良く分割できるかを求める．

2.2 アンサンブル学習

アンサンブル学習は，機械学習の精度を高める工夫の一つである．決定木や線形回帰モデルといったシンプルな機械学習モデル単体では予測能力があまり高くない場合に，複数のモデルを作って多数決をとったり，複数のモデルか

ら得られた結果を足し合わせることで，単体のモデルから得られる予測結果よりも良い予測結果が得られる．このように，複数の弱い学習器 (弱学習器) を組み合わせて予測能力の高い学習器を構成する手法を「アンサンブル学習」と呼ぶ．アンサンブル学習には，弱学習器の組み合わせかたによって大きく分けて 2 種類の方法があり，それぞれ「バギング」,「ブースティング」と呼ばれている．

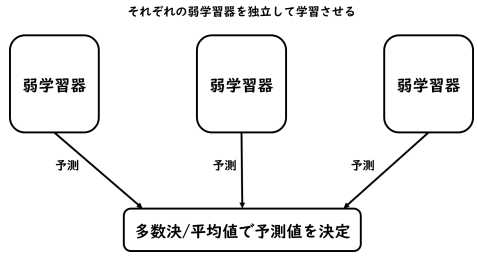

図 2: バギングのイメージ図

バギングは図 2 に示すように，複数の弱学習器を並列に学習させ，予測値を求める際は回帰であれば各弱学習器の予測値の平均，分類問題であれば各弱学習器の予測値の多数決によって決める．学習を並列に実行できて高速であるため，実用的なデータ分析で広く用いられる．特に弱学習器として決定木を用いる場合は「ランダムフォレスト」と呼ばれる非常に有名な手法である．

図 3: ブースティングのイメージ図

ブースティングは図 3 に示すように，予測値の誤差がどんどん小さくなるように弱学習器を直列に増やしていく手法である．最終的な予測値は，各弱学習器の予測値の合計となる．誤差が小さくなるように弱学習器を増やしていくためバギングよりも高い予測性能を得られるが，処理時間の面ではバギングのほうが有利である．ブースティングで得られたモデルは非常に予測性能が高いことから，データ分析のコンペでよく用いられる手法である．

2.3 勾配ブースティング

ブースティングで誤差を小さくする際に勾配降下法を用いる場合を「勾配ブースティング」と呼ぶ．勾配ブースティングによる学習の計算手順を **Algorithm**1 に示す．

subsampling は，学習データの中からランダムに一部を取り出す処理である．モデルの汎化性能を高めるためにこのような処理を行うことも多い．$\beta = 1$ の場合は各学習のループで毎回全てのデータを用いる．また，fit, predict は任意の弱学習器の学習処理と予測処理に相当する．

Algorithm 1 勾配ブースティング (学習)

Require: 説明変数 $X(m \times n$ の行列)，目的変数 $\vec{y}$(長さ m のベクトル)，弱学習器の個数 l，学習率 α，一度の学習で使うデータの割合 β，誤差関数 $J(p)$
Ensure: 学習済の弱学習器 $f_1, \cdots, f_l$
$X', \vec{y'} = \mathsf{subsampling}(X, \vec{y}, \beta)$
$f_1(X) = \mathsf{fit}(X', \vec{y'})$
$\vec{p} = \mathsf{predict}(X')$
$\vec{g} = \alpha \frac{J(p)}{\partial p}$
for $i = 2 \cdots, l$ **do**
  $X', \vec{y'} = \mathsf{subsampling}(X, \vec{y}, \beta)$
  $f_i = \mathsf{fit}(X', \vec{g})$
  $\vec{p} += \mathsf{predict}(f_i, X')$
  $\vec{g} = \alpha \frac{J(p)}{\partial p}$

Algorithm1 に示したように，勾配ブースティングで弱学習器として何を用いるかの制約は無く，誤差関数も自由に設定することができる，

また，勾配ブースティングを用いて学習したモデルでの予測値計算方法を **Algorithm**2 に示す．

Algorithm 2 勾配ブースティング (予測)

Require: 説明変数 $X(m \times n$ の行列), 学習済の弱学習器 $f_1, \cdots, f_l$
Ensure: 予測値 $\vec{p}$
$\vec{p}$ を 0 で初期化 (長さ m)
for $i = 1 \cdots, l$ **do**
  $\vec{p'} = \mathsf{predict}(f_i, X)$
  $\vec{p} = \vec{p} + \vec{p'}$

学習済の全ての弱学習器での予測結果を加算するだけの非常にシンプルな処理である．

勾配ブースティングでは弱学習器として決定木を用いるのが一般的で,「勾配ブースト木 (Gradient Boosted Decision Trees)」と呼ばれる．勾配ブースト木の実装としては XGBoost[2] や LightGBM[3] といった有名なライブラリがある．

有名な機械学習手法としてはディープラーニングもあるが，ディープラーニングでは学習の前に標準化等を用いて説明変数のスケールを揃えておく必要があるのに対し，勾配ブースト木では決定木をベースとしているため，事前に説明変数のスケールを揃える必要が無いというメリットがある．また，予測精度の面でも勾配ブースト木の方が優れていることも多く，近年ではディープラーニングと並んでよく用いられる機械学習手法である．

2.4 評価関数

予測精度の評価関数として，一般的な回帰分析の評価方法である決定係数 R^2 を用いる．学習したパラメータを用いて得られた予測結果を y，ラベルを t，ラベルの平均値を $\overline{t}$ とした時，決定係数 R^2 は下記のように計算できる．M は予測に用いたデータ数である．

$$R^2 = 1 - \frac{\sum_{i=1}^{M}(t_i - y_i)^2}{\sum_{i=1}^{M}(t_i - \overline{y})^2} \quad (1)$$

R^2 は 1 に近いほど予測精度が高いことを示す，

2.5 秘密計算

2.5.1 秘密分散を用いた秘密計算

秘密計算にはいくつか方式があり，中でも秘密情報を「シェア」という複数の断片に変換する秘密分散方式は，データの処理単位が小さく，処理が高速である [1], [14]. 本稿では，n 個のシェアを生成し，k 個以上のシェアからは秘密が復元できるが，k 個未満のシェアからは秘密の情報が全く漏れない (k, n) 閾値法という秘密分散方式を用いる．平文とシェア (暗号文) を区別するため，a の暗号文は $[\![a]\!]$ と書き，括弧がついていないものは平文とする．

例えば 2 つの暗号文 $[\![a]\!], [\![b]\!]$ の加算，減算，乗算は $[\![a]\!] + [\![b]\!], [\![a]\!] - [\![b]\!], [\![a]\!] \times [\![b]\!]$ のように書き，入力がベクトル $[\![\vec{a}]\!], [\![\vec{b}]\!]$ の場合にも同じ記法とする．

2.5.2 プログラマブルな秘密計算ライブラリ MEVAL

MEVAL 我々が開発する秘密分散ベースの秘密計算ライブラリで，加減乗算などの基本的な演算に加えて，公開値除算，除数秘匿除算，指数といった実数演算など 100 以上の演算を組み合わせて自由にプログラムできる [13], [15]. 本稿の提案アルゴリズムは MEVAL 以外の秘密計算ライブラリにも適用できるが，実装では MEVAL を用いている．

3. 提案手法

3.1 秘密計算勾配ブースト木

秘密計算勾配ブースト木のアルゴリズムを **Algorithm**3 に示す．弱学習器である秘密計算決定木は既存手法を用いるため，本稿では秘密計算による決定木の学習や予測処理についての説明は割愛する．誤差関数は二乗誤差を用いた．

秘密計算勾配ブースト木は以下の情報を暗号化したまま計算している．

- 学習データ (説明変数 X，目的変数 $\vec{y}$)
- 個々の木の情報 $f(X)$(木の形，分岐の条件)
- 予測値 p，勾配 g

また，以下の情報は公開値として扱っている．

- 学習データの件数 m
- 説明変数の数 n
- 木の数 l
- 学習率 $\alpha (0 < \alpha < 1)$
- 一度の学習で使うデータの割合 $\beta (0 < \beta < 1)$

subsampling は秘密計算によるシャッフルを行ったのち，

Algorithm 3 秘密計算勾配ブースト木 (学習)

Require: 説明変数 $[\![X]\!](m \times n$ の行列)，目的変数 $[\![\vec{y}]\!]$(長さ m のベクトル)，弱学習器の個数 l，学習率 α，一度の学習で使うデータの割合 β
Ensure: 学習済の弱学習器 $[\![f_1(X)]\!], \cdots, [\![f_l(x)]\!]$
1: $[\![X']\!], [\![\vec{y'}]\!] = \mathsf{subsampling}([\![X]\!], [\![\vec{y}]\!], \beta)$
2: $[\![f_1]\!] = \mathsf{fit}([\![X']\!], [\![\vec{y'}]\!])$
3: $[\![\vec{p}]\!] = \mathsf{predict}([\![f_1]\!], [\![X']\!])$
4: $[\![\vec{g}]\!] = [\![\vec{y'}]\!] - [\![\vec{p}]\!]$
5: $[\![\vec{g}]\!] = \alpha \times [\![\vec{g}]\!]$
6: **for** $i = 2 \cdots, l$ **do**
7: $\quad [\![X']\!], [\![\vec{y'}]\!] = \mathsf{subsampling}([\![X]\!], [\![\vec{y}]\!], \beta)$
8: $\quad [\![f_i]\!] = \mathsf{fit}([\![X']\!], [\![\vec{g}]\!])$
9: $\quad [\![\vec{p'}]\!] = \mathsf{predict}([\![f_i]\!], [\![X']\!])$
10: $\quad [\![\vec{p}]\!] = [\![\vec{p}]\!] + [\![\vec{p'}]\!]$
11: $\quad [\![\vec{g}]\!] = [\![\vec{y'}]\!] - [\![\vec{p}]\!]$
12: $\quad [\![\vec{g}]\!] = \alpha \times [\![\vec{g}]\!]$

データ数 $m \times \beta$ 件を切り出す処理とした．また，小数値 α とのかけ算は (2)〜(4) のように公開値除算を用いることで実現している．k は適当な正の整数で，α として一般的に 0.1 程度の値が用いられることから本稿では $k = 2$ とした．

$$\alpha' = \alpha \times 10^k \quad (2)$$

$$[\![\vec{g}]\!] = \alpha' \times [\![\vec{g}]\!] \quad (3)$$

$$[\![\vec{g}]\!] = [\![\vec{g}/10^k]\!] \quad (4)$$

また，秘密計算による勾配ブースト木の予測アルゴリズムを **Algorithm**4 に示す．

Algorithm 4 秘密計算勾配ブースト木 (予測)

Require: 説明変数 $[\![X]\!](m \times n$ の行列)，学習済の弱学習器 $[\![f_1]\!], \cdots, [\![f_l]\!]$
Ensure: 予測値 $[\![\vec{p}]\!]$
1: $[\![\vec{p}]\!]$ を $[\![0]\!]$ で初期化 (長さ m)
2: **for** $i = 1 \cdots, l$ **do**
3: $\quad [\![\vec{p'}]\!] = \mathsf{predict}([\![f_i]\!], [\![X]\!])$
4: $\quad [\![\vec{p}]\!] = [\![\vec{p}]\!] + [\![\vec{p'}]\!]$

4. 実験

4.1 実験環境

秘密計算サーバとして表 1 に示すマシン 3 台を用いて実験を行った．

表 1: 測定環境

OS	CentOS Linux release 7.3.1611
CPU	Intel Xeon Gold 6144k(3.50GHz 8 コア/16 スレッド) × 2
メモリ	768GB
NW	Intel Ethernet Controller X710/X557-AT 10G リング構成

4.2 処理時間の評価

4.2.1 勾配ブースト木の学習

Algorithm3 に示した提案手法を用いて，入力のデータ数を 100 件〜10 万件として処理時間を測定した結果を表 2 に示す．木の数は 5，木の深さの最大値は 5，データの属性数は 10 とした．またサブサンプルの割合 $\beta = 1$ とし，学習の各ループでは毎回全データを用いた．

表 2: 秘密計算勾配ブースト木 (学習) 処理時間 [s]

100 件	1000 件	1 万件	10 万件
13	15	23	123

1 万件なら約 20 秒，10 万件程度の大規模データでも約 2 分で処理できており，実用的な性能を示した．同じデータセットに対して XGBoost を用いた平文での実験では 100 件で 5[ms]，10 万件で 64[ms] であった．木の数 (num_boost_round) と木の深さ (max_depth) 以外のハイパーパラメータは XGBoost のデフォルト値を用いた．

また属性数を 10〜30 として処理時間を測定した結果を図 4，木の深さを 3〜10 として処理時間を測定した結果を図 5 に示す．属性数を変えた実験での木の数は 5，木の深さの最大値は 5，データ数は 100 とし，木の深さを変えた実験での木の数は 5，データ数は 100，属性数は 10 とした．

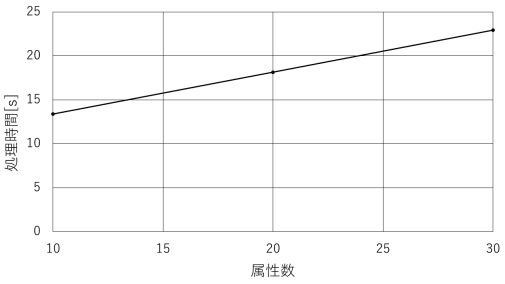

図 4: 属性数を変えた場合の処理時間 [s]

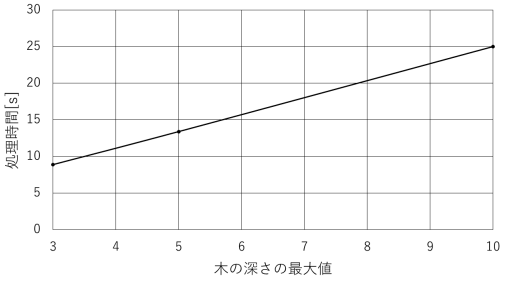

図 5: 木の深さを変えた場合の処理時間 [s]

図 4 および図 5 の結果より，処理時間は属性数や木の深さに比例して増加することが確認できた．

4.2.2 勾配ブースト木の予測

Algorithm4 に示した勾配ブースト木の予測アルゴリズムに対して，100 件〜10 万件のデータで処理時間を測定した結果を表 3 に示す．予測に用いたモデルの大きさは表 2 に示した学習のときと同じである．

表 3: 秘密計算勾配ブースト木 (予測) 処理時間 [s]

100 件	1000 件	1 万件	10 万件
1.6	1.6	2.5	8.6

10 万件でも 8.6 秒と高速に処理できており，予測も実用的な処理時間でできることが確認できた．

4.3 実データでの評価

Boston データセットを用いて処理時間とループ 1 回ごとの決定係数を確認した結果を図 4 に示す．木は全部で 5 つ作成し，各木の最大の深さは 5 としたところ，処理時間は 16 秒であった．

表 4: 各ステップでの決定係数

1 回目	2 回目	3 回目	4 回目	5 回目
0.919	0.924	0.928	0.932	0.935

木を 1 つ追加するごとに決定係数が改善しており，ブースティングが上手くできていることが確認できた．また，XGBoost を用いて同様の設定で測定したところ，処理時間は 6[ms] であった．木の数 (num_boost_round) と木の深さ (max_depth) 以外のハイパーパラメータは，XGBoost のデフォルトの設定を用いた．

同様のデータを用いて筆者らが以前提案した秘密計算ディープラーニング [6] と予測精度 (決定係数) を比較したところ，秘密計算ディープラーニングでは約 640 秒かけて決定係数が 0.9028 だったのに対して，表 4 の結果では 16 秒で決定係数が 0.935 となった．秘密計算ディープラーニングと比較して 40 分の 1 の時間で高い予測精度を得ることができ，秘密計算勾配ブースト木が非常に有用であることか確認できた．

5. おわりに

本稿の貢献は以下である．

- 秘密計算による勾配ブースト木の学習・予測を実用的な処理速度で実現した
- 非常に高い予測精度を持つ AI を秘密計算上でも作れるようになった

秘密計算ディープラーニングでは 640 秒かかって得られた予測精度よりも高い精度を，本稿の秘密計算勾配ブースト木では約 16 秒で達成した．したがって，秘密計算ででき

る機械学習手法が単純に増えただけでなく，勾配ブースト木のような高性能な機械学習手法が扱えるようになったことで，秘密計算 AI 全体の実用性が更に向上したといえる．

参考文献

[1] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *CCS*, pp. 805–817, 2016.

[2] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, Vol. 1, No. 4, pp. 1–4, 2015.

[3] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, Vol. 30, pp. 3146–3154, 2017.

[4] Chester Leung, Andrew Law, and Octavian Sima. Towards privacypreserving collaborative gradient boosted decision trees. Technical report, UC Berkeley, Tech. Rep, 2019.

[5] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. Federated learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, Vol. 13, No. 3, pp. 1–207, 2019.

[6] 三品気吹, 濱田浩気, 五十嵐大. 実用的な秘密計算ディープラーニングの実現. In *CSS*, 2019.

[7] 三品気吹, 濱田浩気, 五十嵐大. 秘密計算ディープラーニングは速いだけでは使えない. In *SCIS*, 2020.

[8] 三品気吹, 深見匠, 濱田浩気, 五十嵐大, 菊池亮. 秘密計算によるプライバシー保護生存時間解析. In *CSS*, 2020.

[9] 三品気吹, 濱田浩気, 五十嵐大, 菊池亮. 秘密実数演算を用いた高速かつ高精度なロジスティック回帰とデータ標準化. In *CSS*, 2020.

[10] 三品気吹, 濱田浩気, 五十嵐大, 菊池亮. 秘密計算ディープラーニングの学習や推論が記述可能な python ライブラリの設計と実装. In *SCIS*, 2021.

[11] 市川敦謙, 須藤弘貴, 竹之内大地, 菊池亮, 濱田浩気, 五十嵐大. 秘密計算において高速な初等関数が機械学習や高度な統計にもたらす実用性. In *SCIS*, 2020.

[12] 深見匠, 三品気吹, 五十嵐大. 秘密計算による正則化線形回帰分析. In *CSS*, 2020.

[13] 五十嵐大. 秘密計算 ai の実装に向けた秘密実数演算群の設計と実装-o(p\) ビット通信量 o(1) ラウンドの実数向け右シフト-. In *CSS*, 2019.

[14] 五十嵐大, 濱田浩気, 菊池亮, 千田浩司. 超高速秘密計算ソートの設計と実装：秘密計算がスクリプト言語に並んだ日. In *CSS*, 2017.

[15] 桐淵直人, 五十嵐大, 濱田浩気, 菊池亮. プログラマブルな秘密計算ライブラリ MEVAL3. *SCIS*, 2018.